Using I/O on Cray XT Systems

Lonnie Crosby
lcrosby1@utk.edu

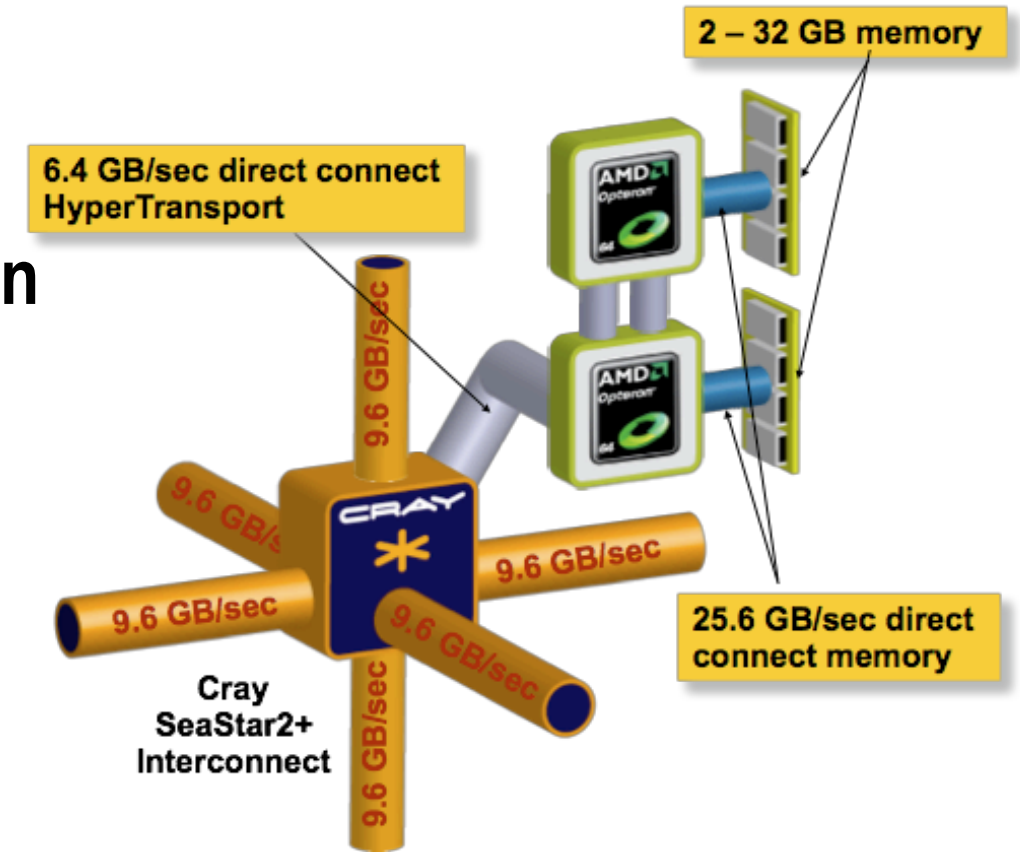
Glenn Brook
glenn-brook@tennessee.edu

NICS Scientific Computing Group

NERSC/OLCF/NICS
Joint Cray XT5 Workshop
February 1 – 3, 2010

Application Performance

- Computation (FLOPs)
  - Processor
- Inter-process Communication
  - Interconnect
- Memory
  - Capacity and Speed
- I/O
  - File System

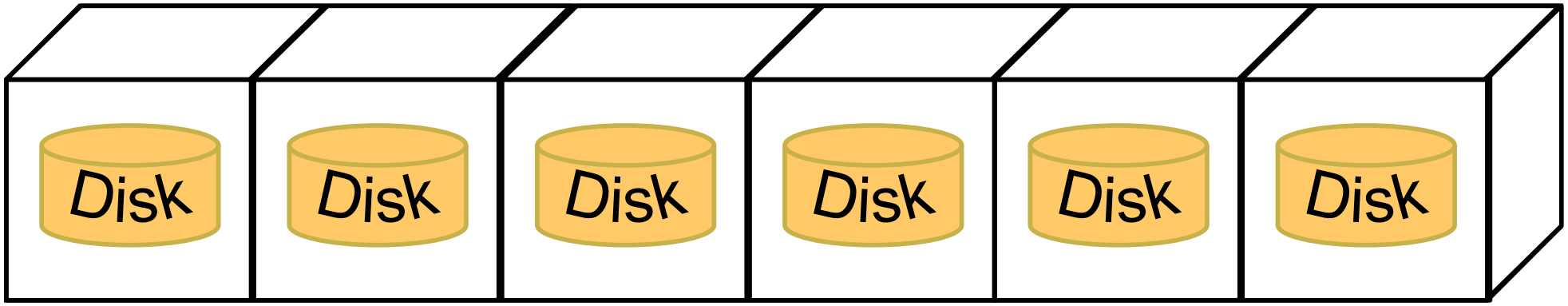


Factors which affect I/O.

- I/O is simply data migration.
  - Memory ⟷ Disk
    - Layout (contiguous?)
- Size of write/read operations
  - Bandwidth vs. Latency
- Number of processes performing I/O
  - I/O Pattern
- Characteristics of the file system
  - Distributed or Shared
- Interaction between processes and file system.

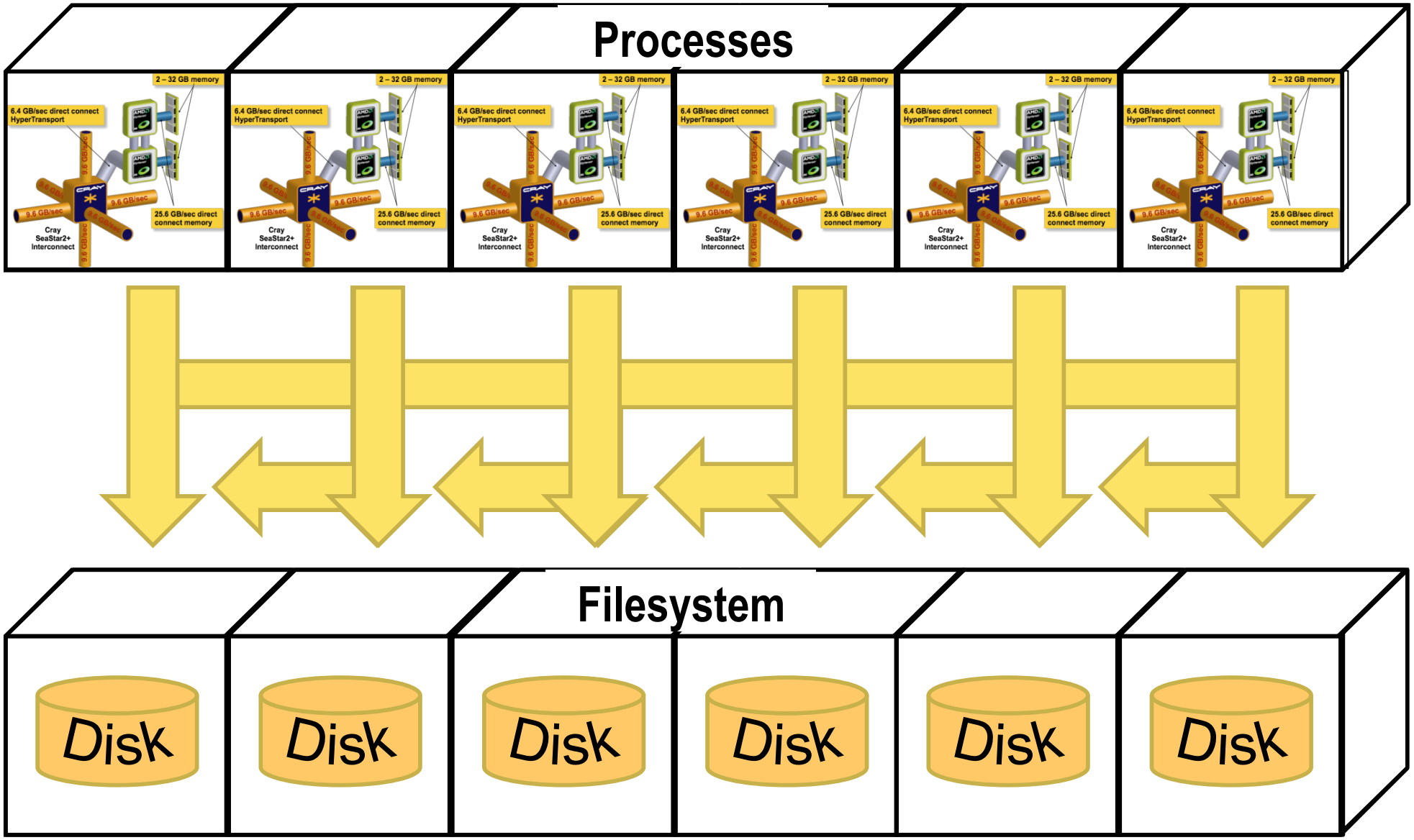
Parallelism

- Process level parallelism
  - MPI
  - IO Libraries (MPI-IO, HDF5, p-netCDF)
- File System parallelism
  - Distributed File System
  - Shared Parallel File System



NICS

Parallelism



Application I/O Patterns

Serial I/O

- Spokesperson
  - One process performs I/O.

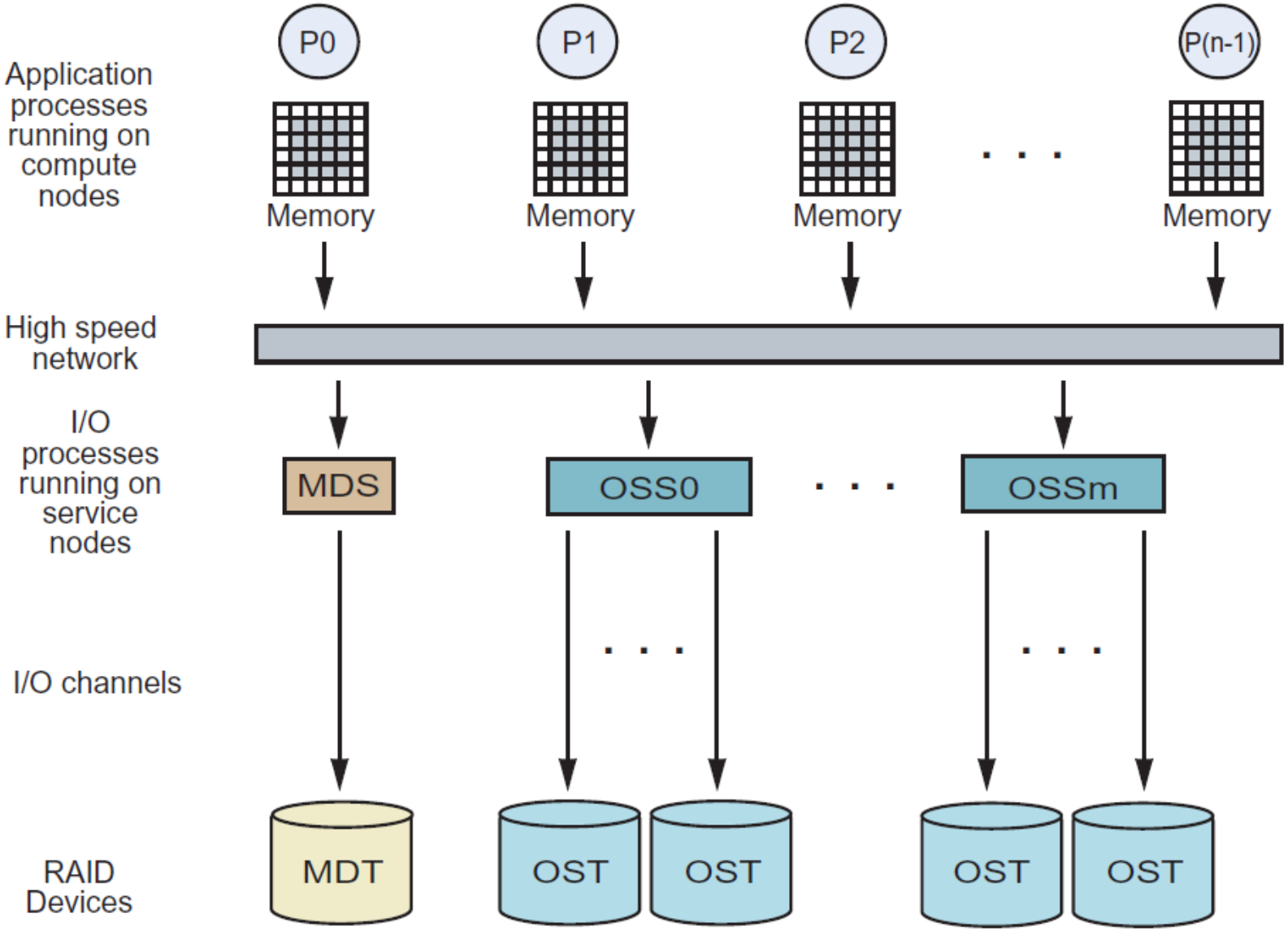
Parallel I/O

- File per Process
  - Each process performs I/O to a single file.
- Single Shared File
  - Each process performs I/O to a single file which is shared.
- Multiple Shared Files
  - Groups of processes perform I/O to a single shared file.

What breaks parallelism in I/O?

- Serial I/O
  - Is limited by the single process which performs I/O.
  - Unable to take advantage of process or file system parallelism
- Parallel I/O
  - Is limited by contention for file system resources.
  - I/O pattern is important in determining the extent of contention

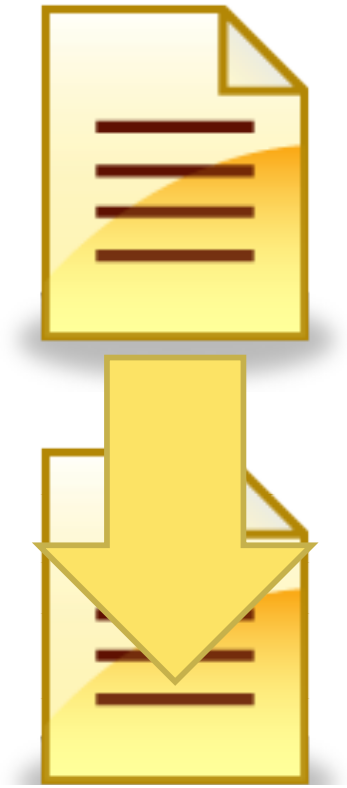
A Bigger Picture: Lustre File System



File Striping: Parallelism for files

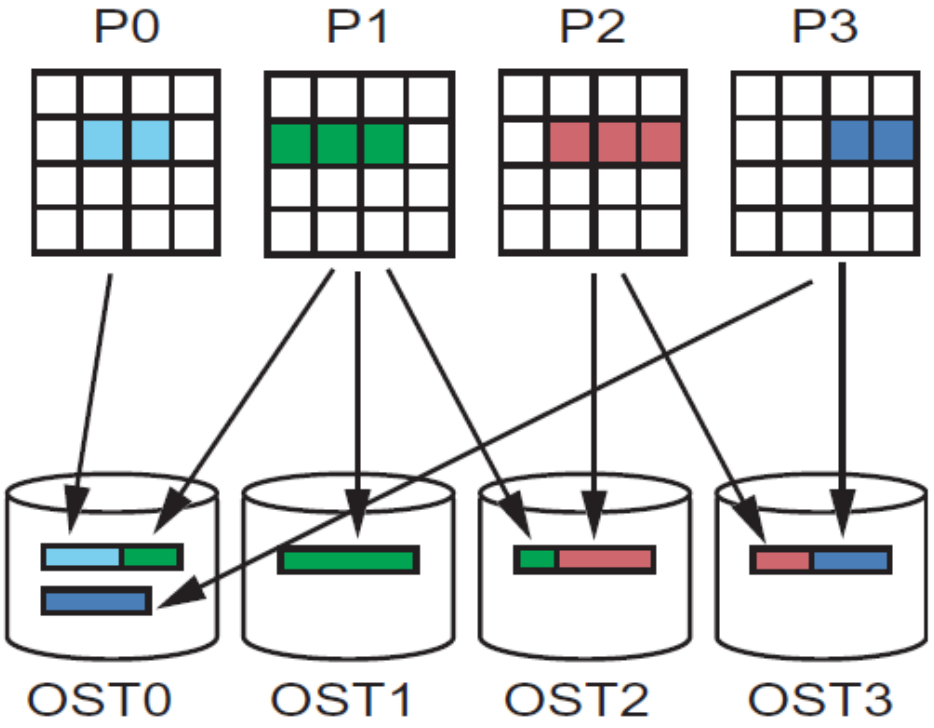
- lfs setstripe
  - Stripe size -s (default: 1M)
  - Stripe count -c 5 (default 4, -1 All)
  - Stripe index -i 0 (default: -1 round robin)

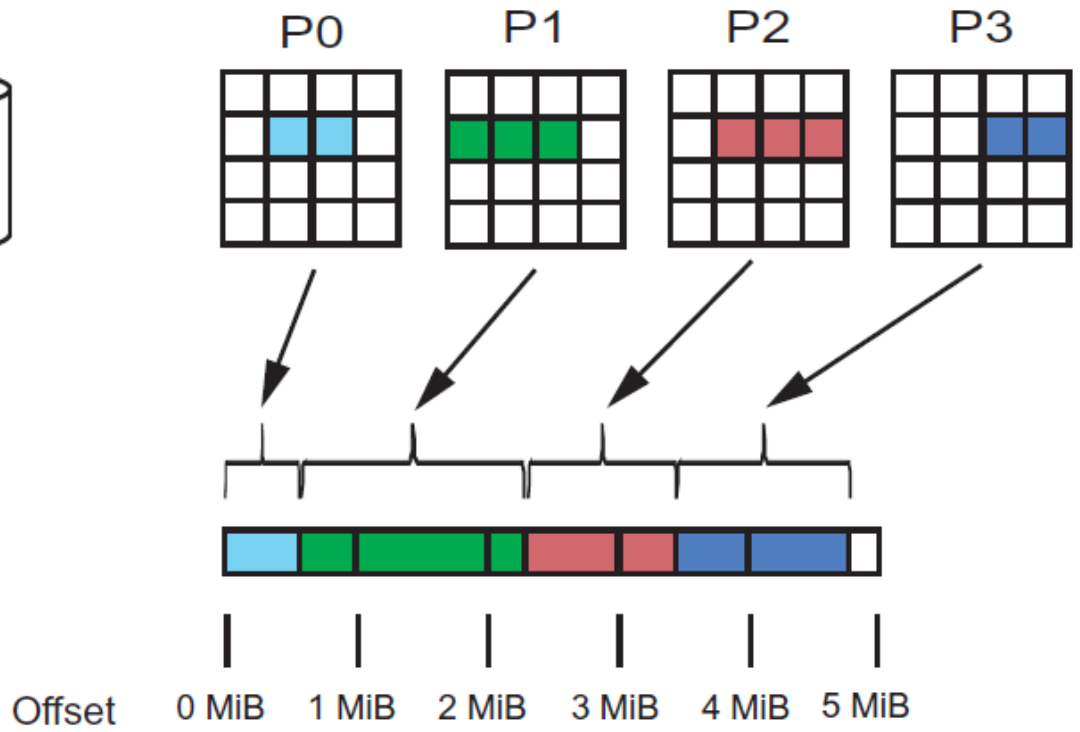
  < file | directory >



NICS

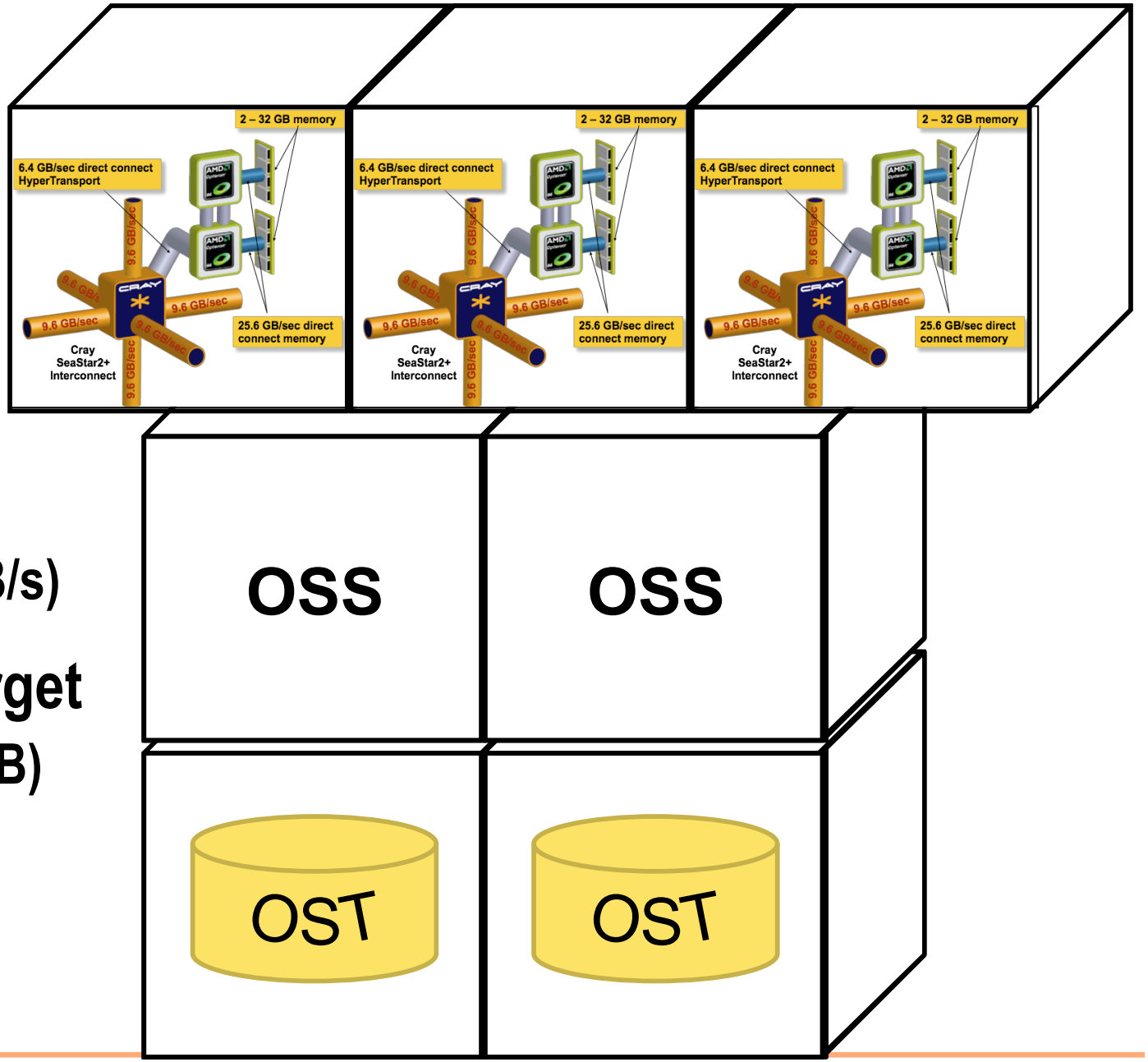
File Striping: Physical and Logical Views





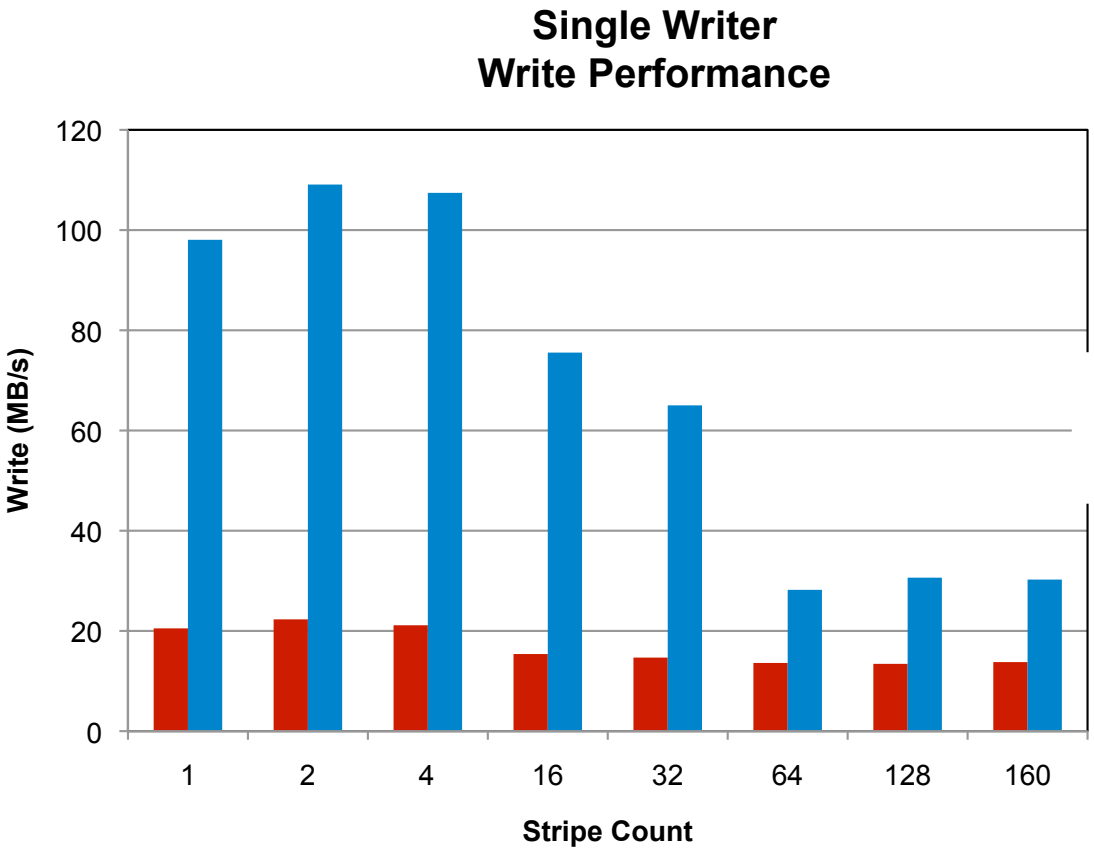
A Bigger Picture

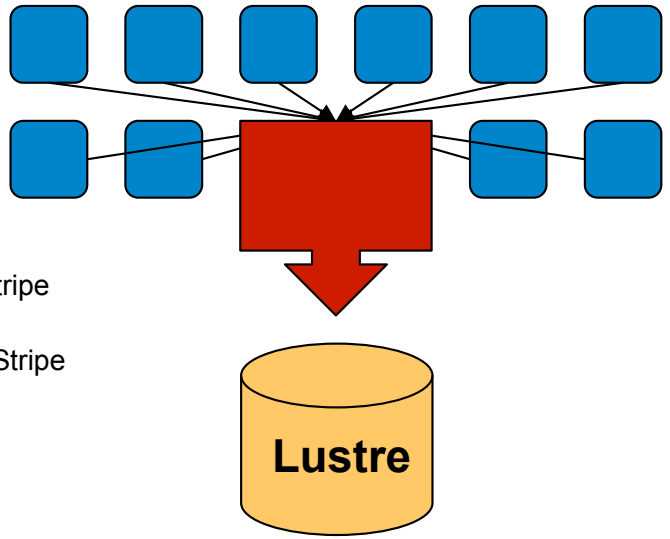
- Computational Nodes
  - Kraken: 8253
- Object Storage Server Nodes
  - Kraken: 48 (30 GB/s)
- Object Storage Target
  - Kraken: 336 (2.4 PB) [7.2 TB Disk]



Spokesperson – Serial I/O

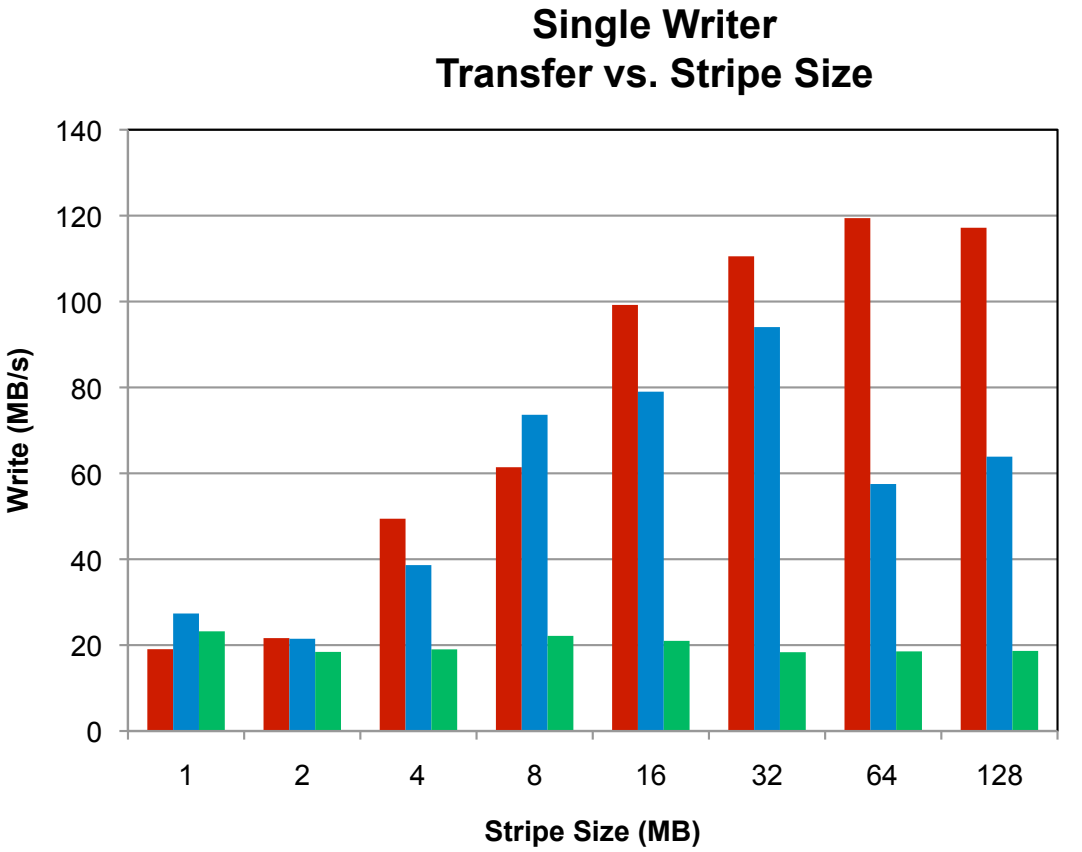
- 32 MB per OST (32 MB – 5 GB) and 32 MB Transfer Size
  - Unable to take advantage of file system parallelism
  - Access to multiple disks adds overhead which hurts performance

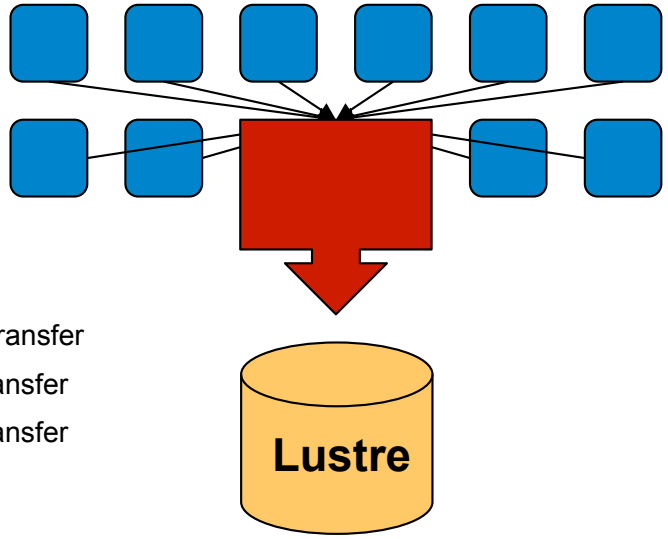




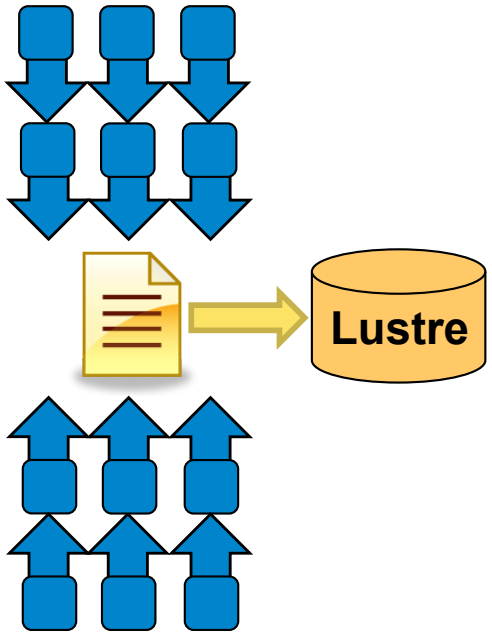
Spokesperson – Serial I/O

- Single OST, 256 MB File Size
  - Performance can be limited by the process (transfer size) or file system (stripe size)





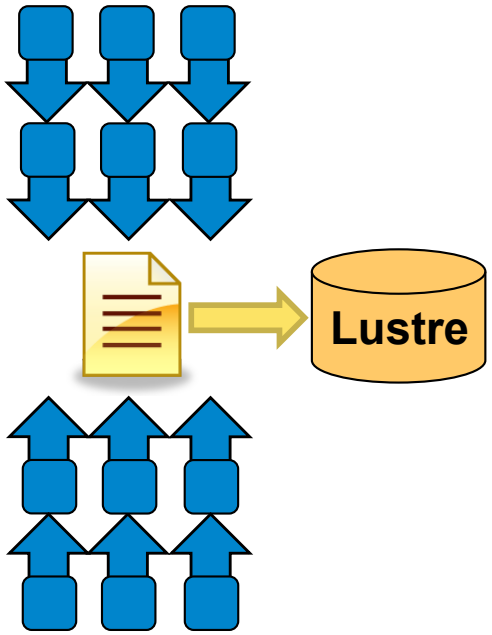
Single Shared File: File Structure

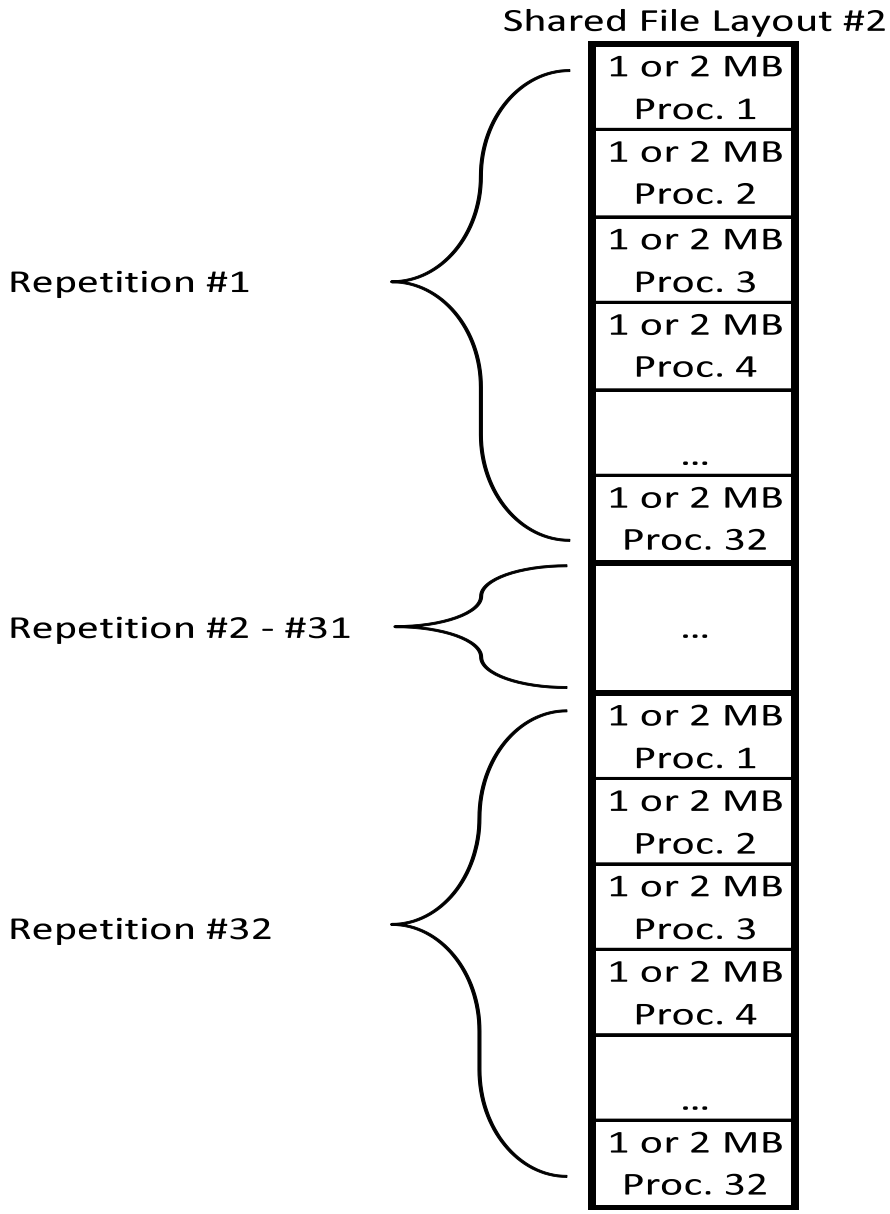


Shared File Layout #1

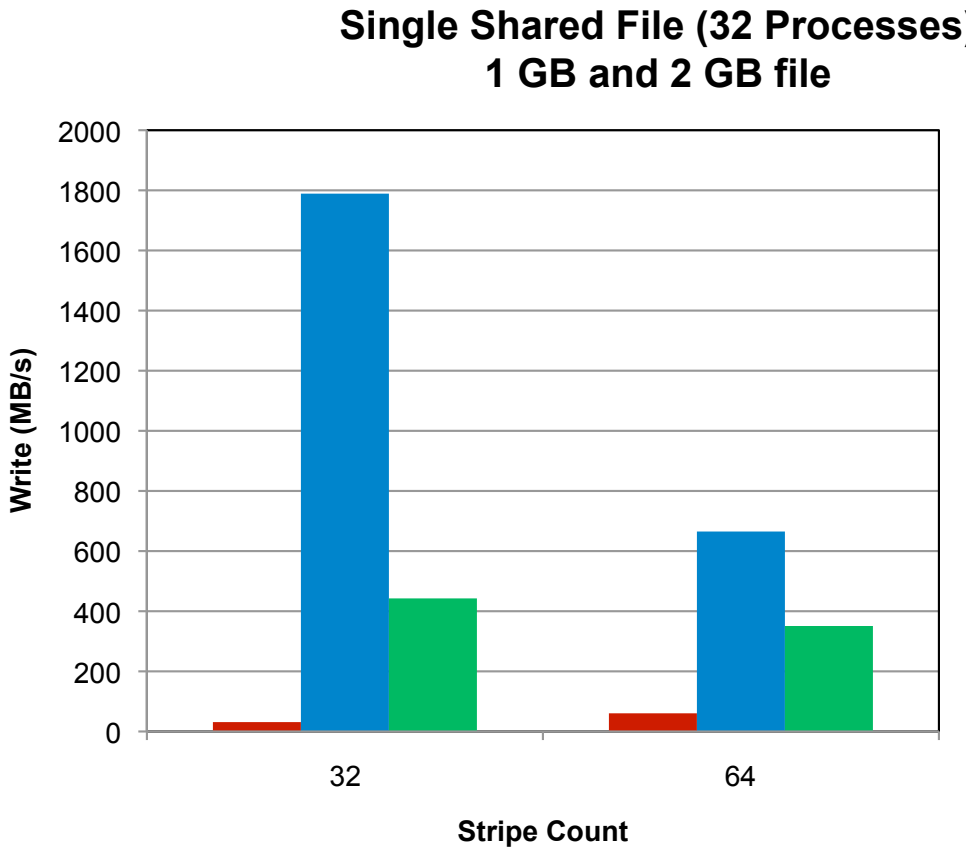
32 or 64 MB Proc. 1
32 or 64 MB Proc. 2
32 or 64 MB Proc. 3
32 or 64 MB Proc. 4
…
32 or 64 MB Proc. 32

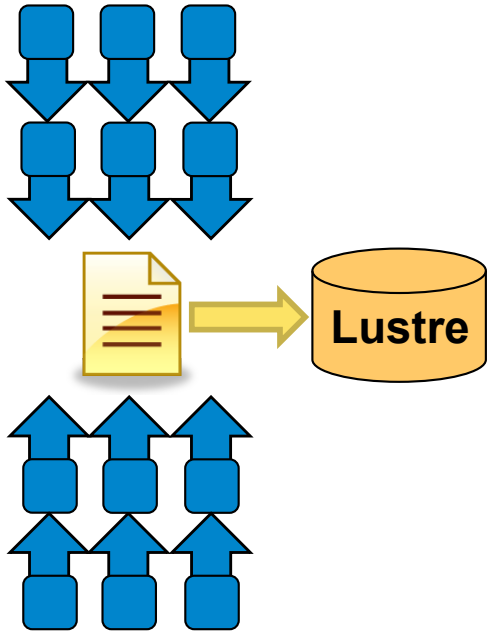
Single Shared File: File Structure





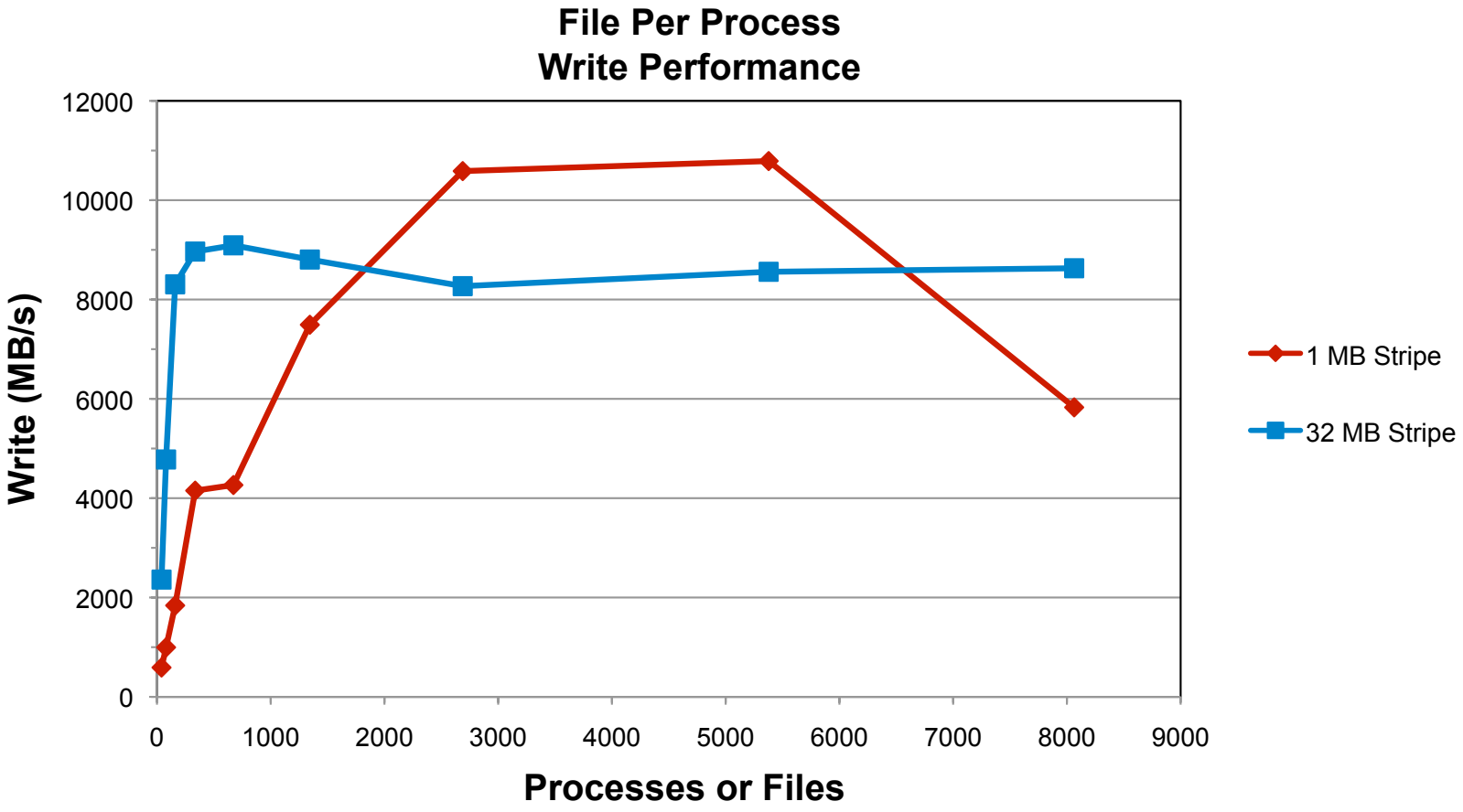
Single Shared File





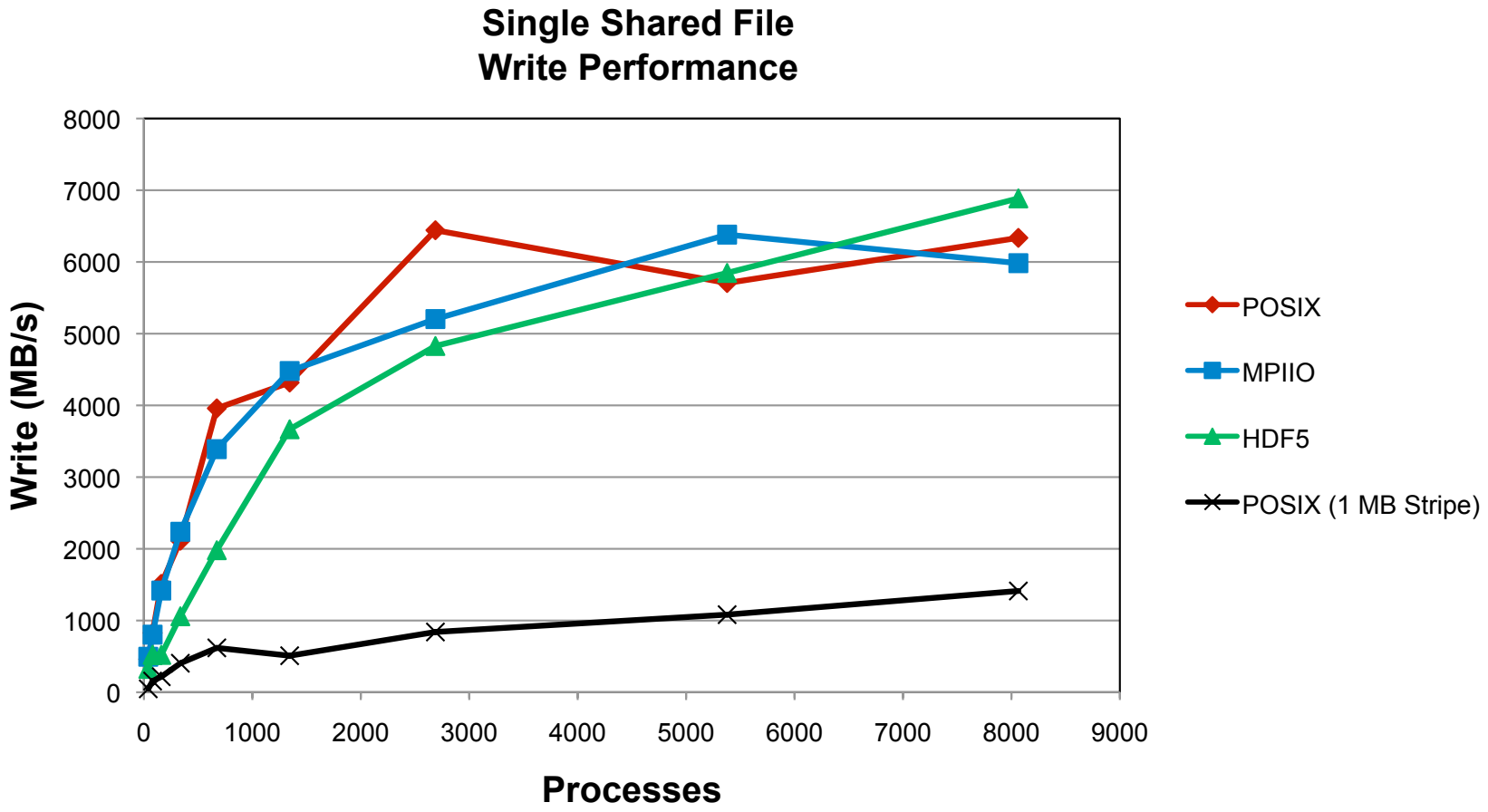
Scalability: File Per Process

- 128 MB per file and a 32 MB Transfer size



Scalability: Single Shared File

- 32 MB per process, 32 MB Transfer size and Stripe size

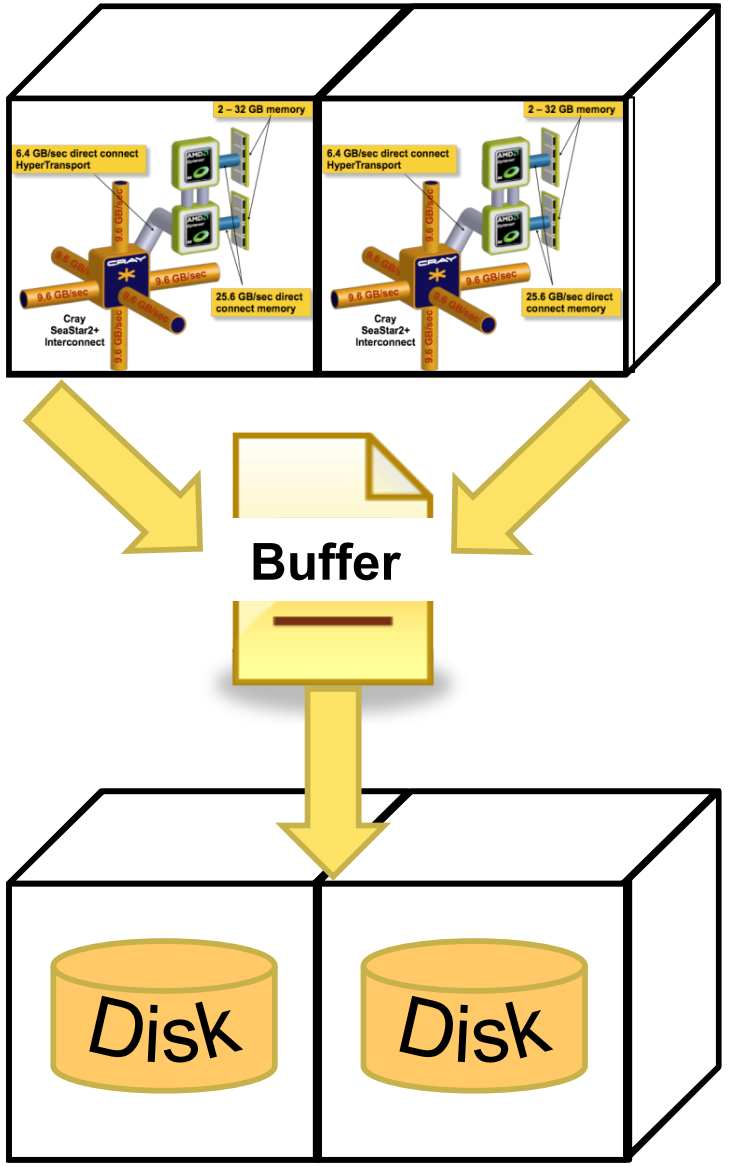


Scalability

- Serial I/O
  - Is not scalable. Limited by single process which performs I/O.
- File per Process
  - Limited at large process/file counts by:
    - Metadata Operations
    - File System Contention
- Single Shared File
  - Limited at large process counts by file system contention.
  - File striping limitation of 160 OSTs in Lustre

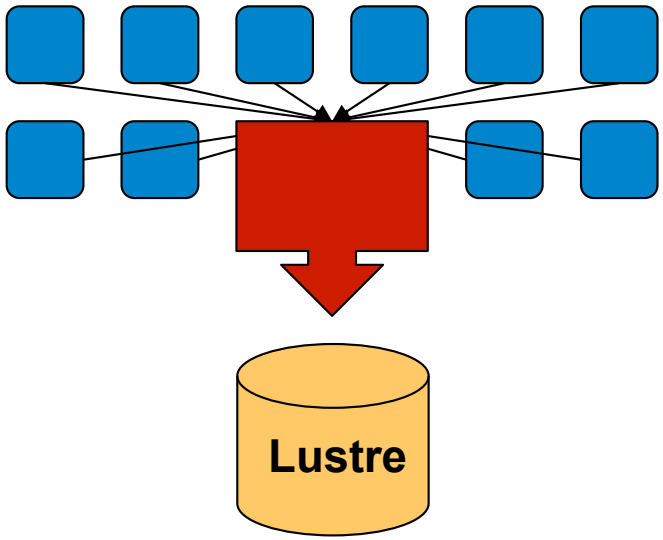
Buffered I/O

- Advantages
  - Aggregates smaller read/write operations into larger operations.
  - Examples: OS Kernel Buffer, MPI-IO Collective Buffering
- Disadvantages
  - Requires additional memory for the buffer.
  - Can tend to serialize I/O.
- Caution
  - Frequent buffer flushes can adversely affect performance.



Standard Output and Error

- Standard Ouput and Error streams are effectively serial I/O.
- Generally, the MPI launcher will aggregate these requests. (Example: mpirun, mpiexec, aprun, ibrun, etc..)
- Disable debugging messages when running in production mode.
  - "Hello, I'm task 32000!"
  - "Task 64000, made it through loop."



Binary Files and Endianess

- Writing a big-endian binary file with compiler flag byteswapio

```
File “XXXXXX"
                  Calls        Megabytes       Avg Size
  Open                1
  Write         5918150      23071.28062           4088
  Close               1
  Total         5918152      23071.28062           4088
```

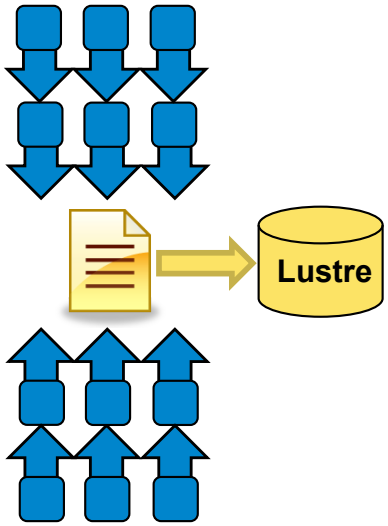
- Writing a little-endian binary

```
File “XXXXXX"
                  Calls        Megabytes       Avg Size
  Open                1
  Write             350      23071.28062       69120000
  Close               1
  Total             352      23071.28062       69120000
```

- Can use more portable file formats such as HDF5, NetCDF, or MPI-IO.

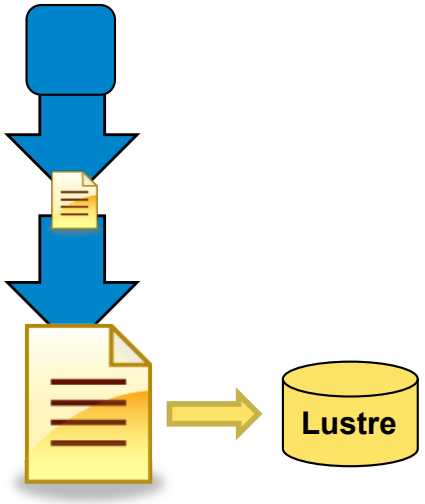
Case Study: Parallel I/O

- A particular code both reads and writes a 377 GB file. Runs on 6000 cores.
  - Total I/O volume (reads and writes) is 850 GB.
  - Utilizes parallel HDF5
- Default Stripe settings: count 4, size 1M, index -1.
  - 1800 s run time (~ 30 minutes)
- Stripe settings: count -1, size 1M, index -1.
  - 625 s run time (~ 10 minutes)
- Results
  - 66% decrease in run time.



Case Study: Buffered I/O

- A post processing application writes a 1GB file.
- This occurs from one writer, but occurs in many small write operations.
  - Takes 1080 s (~ 18 minutes) to complete.
- IOBUF was utilized to intercept these writes with 64 MB buffers.
  - Takes 4.5 s to complete. A 99.6% reduction in time.




```
File "ssef_cn_2008052600f000"
                  Calls          Seconds       Megabytes   Megabytes/sec   Avg Size
  Open                1         0.001119
  Read              217         0.247026        0.105957        0.428931        512
  Write         2083634         1.453222     1017.398927      700.098632        512
  Close               1         0.220755
  Total         2083853         1.922122     1017.504884      529.365466        512
  Sys Read            6         0.655251      384.000000      586.035160   67108864
  Sys Write          17         3.848807     1081.145508      280.904052   66686072
  Buffers used              4  (256 MB)
  Prefetches                6
  Preflushes               15
```

IOBUF -Beta Library

- module load iobuf/beta
- Relink application with the Cray wrappers (ftn, cc, CC)
- Controlled by environmental variable at runtime.
  - setenv IOBUF_PARAMS '*:verbose'
  - man iobuf for more information
- Intercepts standard I/O calls. May not operate with the use of I/O libraries such as netcdf.

MPI-IO Usage

- Included in the Cray MPT library.
- Environmental variable used to help MPI-IO optimize I/O performance.
  - setenv MPICH_MPIIO_HINTS
  - man mpi for more information
- If given appropriate information (stripe count, size) can choose aggregators in collective operations that are Lustre stripe aligned. (collective buffering).

Conclusions

- Serial I/O
  - For a single process, performance is limited by the single I/O stream.
  - For the file-per-process pattern, the limitation is due to simultaneous metadata operations (file open) at large core counts. Additionally, increasing contention for file system resources can adversely affect performance.
- Parallel I/O
  - For a single, shared file, the limitation is due to file system contention at large core counts. Lustre limitation of 160 OSTs per file.
  - MPI-IO can be utilized to minimize file system contention at large core counts by utilizing collective buffering and appropriate hints.
- Lustre
  - Appropriate stripe settings should be utilized to minimize file system contention.

Subsetting I/O

- At large core counts, I/O performance can be hindered
  - by the collection of metadata operations (File-per-process) or
  - by file system contention (Single-shared-file).
- One solution is to use a subset of application processes to perform I/O. This limits
  - the number of files (File-per-process) or
  - the number of processes accessing file system resources (Single-shared-file).
- If you can not implement a subsetting approach, try to limit the number of synchronous file opens to reduce the number of requests simultaneously hitting the metadata server.

I/O Best Practices

- Small files (< 1 MB to 1 GB) that are accessed by a single process (serial I/O or file-per-process) should be set to a stripe count of 1.
- Medium sized files (> 1 GB) that are accessed by a single process (serial I/O or file-per-process) should be set to utilize a stripe count of no more than 4 (default).
- Large files (>> 1 GB) should be set to a stripe count that would allow the file to be written to the Lustre file system. The stripe count should be adjusted to a value larger than 4 (default). Such files should never be accessed by a serial I/O or file-per-process I/O pattern.

I/O Best Practices (continued)

- Single shared files should have a stripe count equal to the number of processes. If the number of processes accessing the file is greater than 160 then the stripe count should be set to -1 (max 160).
- Create directories with different stripe settings to control the stripping pattern of included files. Use the "lfs setstripe –c <count> -s <size> <directory>" command to assign a striping pattern to a directory.
- Limit the number of files within a single directory by incorporating additional directory structure (e.g. sqrt(N) directories of sqrt(N) files). Set the Lustre stripe count of such directories which contain many small files to 1.

I/O Best Practices (continued)

- The file-per-process I/O pattern is not scalable to large core counts. Metadata operations become restrictive at process/file counts larger than 5000. This limit is lower if files have a stripe count greater than 1. Limit the number of files by selecting a subset of processes to conduct I/O or changing to a different I/O pattern.
- The single, shared file I/O pattern shows decreasing performance improvements at large core counts. File system contention limits performance at process counts larger than 5000. Limit the number of processes accessing a shared file by selecting a subset to conduct I/O, utilize additional shared files, or utilize I/O libraries such as MPI-IO (collective buffering).

I/O Best Practices (continued)

- Increase the size of I/O write operations to improve performance and align them with the Lustre striping.
- Avoid excessive use of stdout and stderr I/O streams (debugging messages).
- Avoid the use of byteswapio and similar compiler flags.
- Set the Lustre stripe size to allow for better stripe alignment with parallel I/O. Avoid situations in which processes communicate with all utilized OSTs. Take into account the shared file layout, the number of processes, and the size of I/O operations.
- The Lustre stripe index should not be set to a value other than -1.

I/O Best Practices (continued)
(from the NCCS website)

- Open files read-only whenever possible.
  - If the access time on the file does not need to be updated, the open flags should be O_RDONLY | O_NOATIME.
  - If this file is opened by all files in the group, the master process (rank 0) should open it O_RDONLY with the remaining processes (rank > 0) opening it O_RDONLY | O_NOATIME.
- Read/stat small, shared files from a single task and broadcast the data to the remaining tasks.
  - Instead of making a read/stat (and open) request per task, we are making only 1.
  - The broadcast uses a fanout which reduces network traffic by allowing the SeaStar routers of intermediate nodes to process less data.

References

- Lustre File System – White Paper October 2008
  - http://www.sun.com/software/products/lustre/docs/lustrefilesystem_wp.pdf
- Introduction to HDF5
  - http://www.hdfgroup.org/HDF5/doc/H5.intro.html
- The NetCDF Tutorial
  - http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial.pdf
- NICS I/O Tips
  - http://www.nics.tennessee.edu/io-tips
- NCCS Spider (Lustre) Best Practices
  - http://www.nccs.gov/user-support/general-support/file-systems/spider/